Extended Techniques for Flexible Modeling and Execution of Data Mashups

Pascal Hirmer, Peter Reimann, Matthias Wieland and Bernhard Mitschang
Institute of Parallel and Distributed Systems / Application Systems,
University of Stuttgart, Unversitaetsstrasse 38, D-70569 Stuttgart, Germany

Keywords: Data Mashups, Ad-hoc Data Integration, Patterns, Data Flow, Sensor Data.

Abstract: Today, a multitude of highly-connected applications and information systems hold, consume and produce huge amounts of heterogeneous data. The overall amount of data is even expected to dramatically increase in the future. In order to conduct, e.g., data analysis, visualizations or other value-adding scenarios, it is necessary to integrate specific, relevant parts of data into a common source. Due to oftentimes changing environments and dynamic requests, this integration has to support ad-hoc and flexible data processing capabilities. Furthermore, an iterative and explorative *trial-and-error* integration based on different data sources has to be possible. To cope with these requirements, several data mashup platforms have been developed in the past. However, existing solutions are mostly non-extensible, monolithic systems or applications with many limitations regarding the mentioned requirements. In this paper, we introduce an approach that copes with these issues (i) by the introduction of patterns to enable decoupling from implementation details, (ii) by a cloud-ready approach to enable availability and scalability, and (iii) by a high degree of flexibility and extensibility that enables the integration of heterogeneous data as well as dynamic (un-)tethering of data sources. We evaluate our approach using runtime measurements of our prototypical implementation.

1 INTRODUCTION AND BACKGROUND

Data mashups are used for the ad-hoc, flexible access to interesting data sources and for the dynamic, data-driven integration of relevant data (Daniel and Matera, 2014). Due to user requirements, such data mashups have to offer a scalable, tenant-aware solution that supports heterogeneous data, i.e., structured as well as unstructured data. However, existing solutions have several limitations. To analyze their shortcomings, we compared the existing mashup solutions Yahoo Pipes[1], IBM Mashup Center[2], Intel Mashmaker[3] and OpenIoT Mashup[4] regarding the mentioned aspects. We found out that there are several limitations regarding (i) coping with different requirements of various scenarios, i.e., universality of the implementation, (ii) usability by non IT-experts, (iii) handling of heterogeneous – especially unstructured – data, and even if basically supported, (iv) scalability due to an oftentimes complex, monolithic architecture. Furthermore, the handling of *"live data"*, e.g., sensor data streams, is only supported by one of these solutions – OpenIoT Mashup. However, this implementation can cope with sensor data, exclusively. In this paper, we tackle these limitations by an approach that (i) uses patterns to abstract from implementation details, which enables using a tailor-made implementation specific to a respective scenario, (ii) enables usage by non IT-experts by exclusively using means specific to their domain, (iii) enables availability and scalability by providing our solution in the cloud using web technologies and stateless, scalable services, exclusively, (iv) facilitates the integration of various kinds of data through a generic, extensible approach, and (v) enables dynamic (un-)tethering of data sources. To prove the feasibility of our approach, we created a prototype that implements the introduced concepts. Furthermore, an evaluation of our approach is conducted through runtime measurements.

The remainder of this paper is structured as follows: in *Section 2*, the basics of data mashups are described. In *Section 3* to *Section 5*, the main contribution of our paper is presented: we introduce extended techniques for flexible modeling and execution of data mashups. In *Section 6*, we present a prototype of our concepts

[1] http://pipes.yahoo.com/pipes/
[2] http://pic.dhe.ibm.com/infocenter/mashhelp/v3/
[3] http://intel.ly/1BW2crD
[4] http://openiot.eu/

Hirmer P., Reimann P., Wieland M. and Mitschang B..
Extended Techniques for Flexible Modeling and Execution of Data Mashups.
DOI: 10.5220/0005558201110122
In *Proceedings of 4th International Conference on Data Management Technologies and Applications* (DATA-2015), pages 111-122
ISBN: 978-989-758-103-8

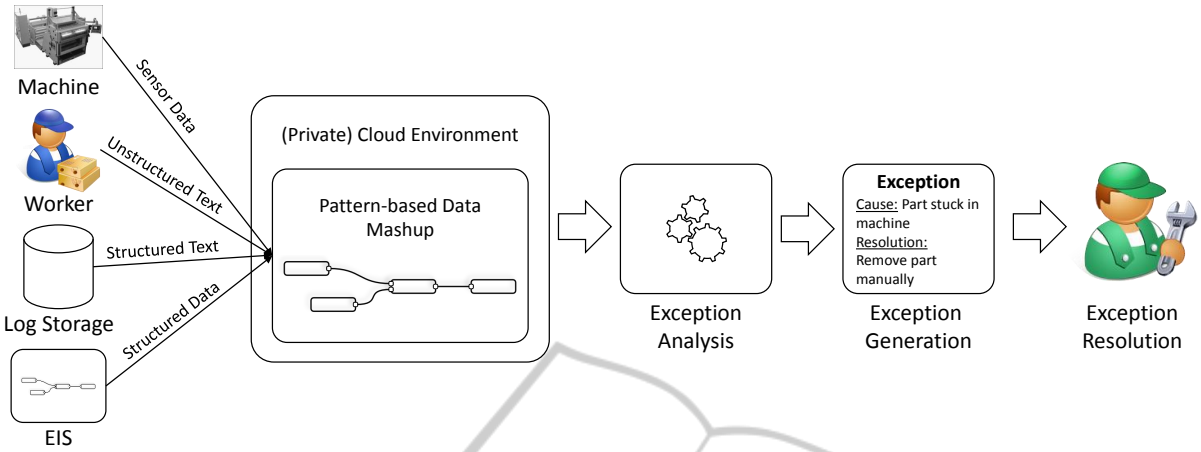


Figure 1: Motivating Scenario based on (Kassner and Mitschang, 2015).

as well as a runtime-based evaluation and in *Section 7*, we describe related work. Finally, we give a summary and present future work in *Section 8*.

1.1 Motivating Scenario

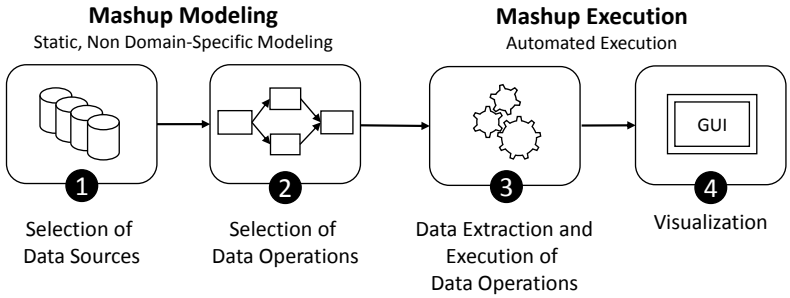
To further clarify our approach, we introduce the following motivating scenario, which is based on (Kassner and Mitschang, 2015) and shown in Figure 1: a car company producing a large amount of cars per day operates a multitude of IT systems that produce heterogeneous data. The data sources in this production environment are: (i) the sensors of the manufacturing machines, (ii) enterprise information systems (EIS) (e.g., to coordinate processes), (iii) structured databases (e.g., containing logs), and (iv) unstructured text e.g., submitted by factory workers. In this production environment, an occurring exception in one of these systems oftentimes leads to a complete production stop, which results in high costs. Solving these exceptions in an efficient manner would reduce these costs severely due to enabling a fast resumption of the production process. However, to realize such an efficient exception resolution, we need a means to automatically recognize occurring exceptions and to initiate their solution. For example, a damaged machine could be recognized based on the machine's log data, its sensor data and additional textual input of a worker describing the error. Once the damage is recognized, a corresponding repair worker can be called automatically to repair it.

To realize this scenario, exceptions have to be recognized based on the factory's data sources. To do so, we need a means to integrate highly heterogeneous data sources and conduct data analysis on the integrated result. Each exception could be recognized based on the integration of different kinds of data sources. As a consequence, static Extract-Transform-Load (ETL) processes should not be used in this scenario. Furthermore, the integration has to cope with a huge amount of data (e.g., sensor data), handle their heterogeneity, i.e., structured and unstructured data, process the integration efficiently, and has to allow an easy adding or removing of data sources to be integrated, depending on which sources are necessary to recognize a specific exception. For example, if the integration of log files and sensor data do not lead to the recognition of an exception, another data source has to be added such as textual input of a worker. As a consequence, ad-hoc processing and flexibility are an important requirement in this scenario. The integrated result is used for data analysis of occurred exceptions. In case no exception has been recognized, additional data sources have to be integrated. The introduced approach in this paper enables such scenarios by providing extended techniques for flexible modeling and execution of data mashups. In the following, we describe our concepts based on this motivating scenario.

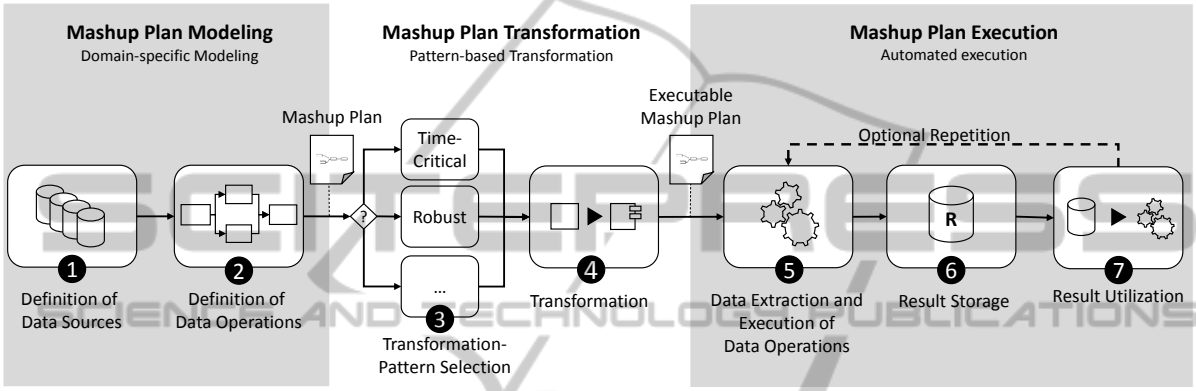
2 CHARACTERISTICS OF DATA MASHUPS

Data mashup platforms aim at enabling the flexible, ad-hoc integration of heterogeneous data sources (Daniel and Matera, 2014). In contrast to application mashups, they focus on the data layer, exclusively. Data mashups are usually defined by the execution of data operations, such as filter or join operations, which extract, alter and integrate data from different sources. The result is a single data source that holds the integrated data. The data to be extracted and *mashed up* is specified by the user before the mashup is executed.

Figure 2(a) displays the typical steps to create and



(a) State of the Art of Mashups



(b) Extended Mashup Approach

Figure 2: Comparison of Traditional Mashup Approaches and our Extended Approach.

execute a data mashup. In the first step, the user defines the data sources to be integrated as well as the specific data to be extracted. In the second step, the data operations for the mashup and their execution order are determined. By doing so, the user specifies how the extracted data should be altered, enhanced, filtered or aggregated to achieve the desired result. After that, the automated steps of the data mashup are processed. In step three, the mashup application receives the user input, extracts the data from the sources and executes the data operations in the specified order. After that, in step four, the result is usually visualized or stored into a suitable data store.

The interested reader is referred to the work of Hoyer et al. (Hoyer et al., 2011) that describes the benefits of data mashups regarding financial, business, operational and user aspects. Furthermore, we recommend the work of Daniel et al. (Daniel and Matera, 2014), summarizing the state of the art of mashups.

3 EXTENDED DATA MASHUP APPROACH

This section contains an overview of the main contribution of our paper, introducing extended techniques for data mashups that enable *"click & run"* for integration scenarios. To enable an effective provisioning of our solution in the cloud as well as usability by domain-experts, we subdivide the approach into three abstraction levels that are shown in Figure 2(b): (i) the modeling level, (ii) the transformation level, and (iii) the execution level. In the context of this paper, a domain-expert is, e.g., a scientist or a business person such as a sales analyst, lacking necessary programming skills or technical knowledge about data integration.

In contrast to traditional data mashups as shown in Figure 2(a), our extended approach (depicted in Figure 2(b)) offers highly-increased flexibility for modeling and executing data mashups. In the first two steps of our extended approach, a domain-expert defines the data sources as well as the data operations to be executed for the mashup, using a domain-specific model called *Mashup Plan*. In step 3, so called *transformation patterns* are selected to choose an implementation suitable for the respective scenario (e.g., time critical or robust). After that, in step 4, the Mashup Plan is transformed into an executable model depending on the transformation pattern that was selected in step 3. In step 5, this executable model is executed using a

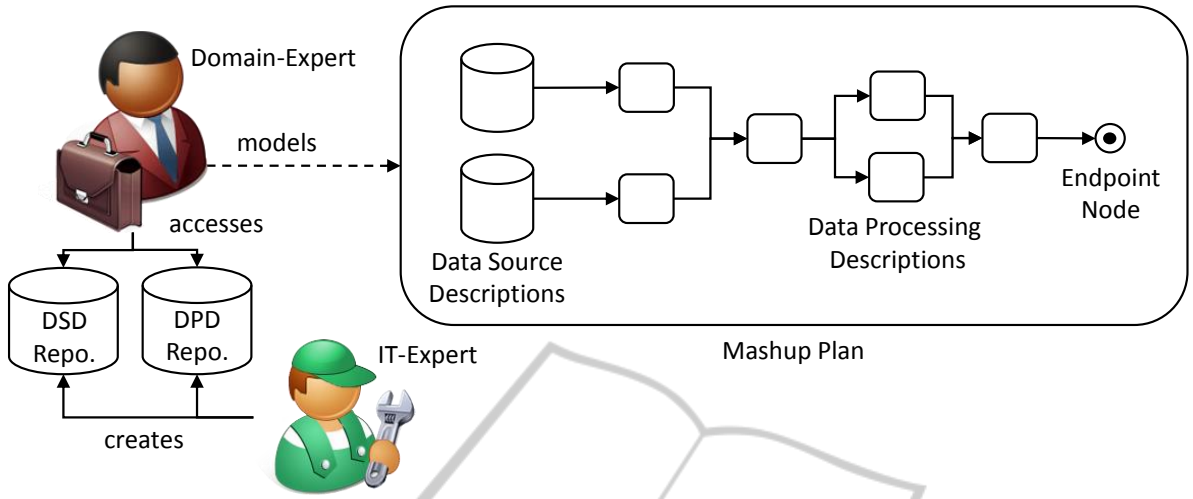


Figure 3: Overview of the Modeling Level.

suitable engine. After execution, the integrated result is stored into a data store (step 6) and it is available for visualization, data analysis or other value-adding scenarios (step 7).

Each of the introduced abstraction levels focuses on one of the main requirements of our solution that were introduced in Section 1. The modeling level, representing the user interface, mainly focuses on the usability issue. Thereby, a domain-specific model is introduced to define data sources and data operations for the mashup in a non-technical manner, i.e., suitable for domain-experts. In addition, transformation patterns are defined to enable choosing a suitable implementation for respective use case scenarios. The transformation pattern *"Time-Critical Mashup"* e.g., ensures an efficient execution of the mashup, whereas the transformation pattern *"Robust Mashup"* refers to an implementation that guarantees some kind of robustness, e.g., regarding stability and communication problems. The transformation level enables the support of various kinds of data by means of a flexible, automatic transformation of the domain-specific model into an executable model. Furthermore, the selection of a corresponding implementation depends on the transformation patterns defined on modeling level. Finally, the execution level mainly focuses on the scalability of our solution. Therefore, all system components are provisioned in a cloud environment. That is, the solution is composed of stateless cloud services, exclusively, which can be automatically provisioned (Binz et al., 2014) as well as scaled and managed, independently.

4 MASHUP PLAN MODELING

The modeling level offers a cloud service to the end user of our approach, i.e., to a domain-expert. The user's goal is to define and execute mashup scenarios by exclusively using means specific to the domain s/he is familiar with. By doing so, all technical and implementation aspects should be abstracted in order to remove the burden to deal with such low-level details from domain-experts. We introduce three concepts at this abstraction level that are displayed in Figure 3: (i) *Data Source Descriptions* (DSD) that describe the data sources to be used for the mashup in a non-technical manner, (ii) domain-specific *Data Processing Descriptions* (DPD), representing data operations (e.g., filter or aggregation operations) to be executed, and (iii) *Mashup Plans*, defining the order of Data Source Descriptions and Data Processing Descriptions. DPDs and DSDs can be arranged as *modeling patterns*, however, this is part of our future work.

4.1 Mashup Plans

As mentioned before, mashups are realized by extracting data from their sources and by executing an arbitrary number of data operations in a predefined order. To enable domain-specific modeling, we introduce *Mashup Plans* that are related to the *Pipes and Filters* (Meunier, 1995) pattern and contain descriptions of the data sources to be integrated as well as an arbitrary number of Data Processing Descriptions, abstracting from fine-grained data operations. Similar to existing work, e.g., of Zweigle et al. (Zweigle et al., 2009), a Mashup Plan is a directed, cohesive

flow graph containing nodes and edges. A node in the Mashup Plan is either a DSD or a DPD and the edges connecting these nodes describe the data and control flow. Mashup Plans are modeled manually by domain-experts. Because of that, it is reasonable to simplify the plan creation by enabling graphical modeling. The format of Mashup Plans is arbitrary, however, using existing languages such as XML or JSON can ease modeling and processing due to available tool support.

We further define the following restrictions for the Mashup Plan: (i) a completely modeled Mashup Plan contains at least one Data Source Description and at least one Data Processing Description, (ii) the modeled Data Source Descriptions represent the starting points of the flow-based Mashup Plan, thus, may only contain outgoing data flow connections, the DPDs may be connected in an arbitrary manner, and (iii) a Mashup Plan has a single output represented by an endpoint node (depicted in Figure 3). The DSDs and DPDs used to model Mashup Plans are stored in corresponding repositories, which can be accessed by the user during modeling (cf. Figure 3).

4.2 DSD Modeling

When modeling Mashup Plans, the user first defines the data sources to be integrated using Data Source Descriptions. These descriptions are predefined and contained in the DSD repository. Each DSD contains information about the data source location and its access information, e.g., a database port, a URL, a sensor API path etc. The data source access information have to be predetermined in the DSD repository by a technical expert, e.g., an IT expert of an enterprise. The DSD repository enables extensibility of these descriptions. Once a sufficient set of Data Source Descriptions is available, usually no more expert interaction is necessary. In addition to the access information, the Data Source Descriptions determine the data to be integrated in a human-readable format tailor-made for domain-experts. This can be accomplished using artifact-centric approaches (Künzle et al., 2011), (Cohn et al., 2009). The data is represented by so-called business artifacts that correspond to business-relevant objects and abstract from the specific data. In the motivating scenario introduced in Section 1, this includes production machines, machine logs, information systems and factory workers. The artifacts manage relevant information about these business objects and about their life cycle in a domain-specific and thus abstract way. The specific data of a machine artifact, e.g., is produced by a multitude of sensors, the artifact of an information system contains structured production process information, and the artifact of a worker incorporates references to a machine, an information system and contains unstructured textual input describing a problem that occurred. These artifacts and their domain-specific descriptions make the modeling of data sources tailor-made to domain-experts. Using mapping approaches, they can be bound to various kinds of underlying data structures – ranging from traditional relational databases to unstructured text data (Sun et al., 2014).

In conclusion, the use of an artifact-centric approach enables domain-experts to model domain-specific business objects (e.g., a specific production machine) in the Mashup Plan without any necessary knowledge of low-level data structures such as sensors, relational databases or unstructured data formats.

4.3 DPD Modeling

Existing mashup and data integration solutions oftentimes require the user to define technical data operations to be executed, including low-level details. This leads to applications that cannot be directly used by domain-experts. To cope with this issue, we introduce abstract, parameterized Data Processing Descriptions, which are specific to the user's domain and describe data operations for the mashup in a non-technical manner. DPDs can be further divided into several abstraction levels as described by Reimann et al. (Reimann et al., 2014) to enable a *separation of concerns*. We describe two examples of such descriptions:

Data Selection DPD. The data selection DPD can be used as a filter in the Mashup Plan. This DPD is connected to exactly one data source and selects specific data from it. Information about the data to be selected is determined by the user during modeling of the Mashup Plan and is attached as a parameter to the DPD. In the motivating scenario, a domain-expert may use this DPD, associate it with artifacts of a production log, and specify it to select relevant entries, exclusively (e.g., within a specific time frame).

Data Combination DPD. The data combination DPD can be used to combine data of two or more sources. The actual implementation of the DPD can be realized by technical data operations, such as data join, union or aggregation, which depend on the kind of data being combined. When modeling the data combination DPD, the artifact-centric models introduced in Section 4.2 are combined. In the motivating scenario for example, data from machine-, log- and worker-artifacts may be grouped according to a specific failure type or time frame.

The introduced DPDs and artifact-centric DSDs represent domain-specific means for the Mashup Plan modeling, which lead to high usability and it allows

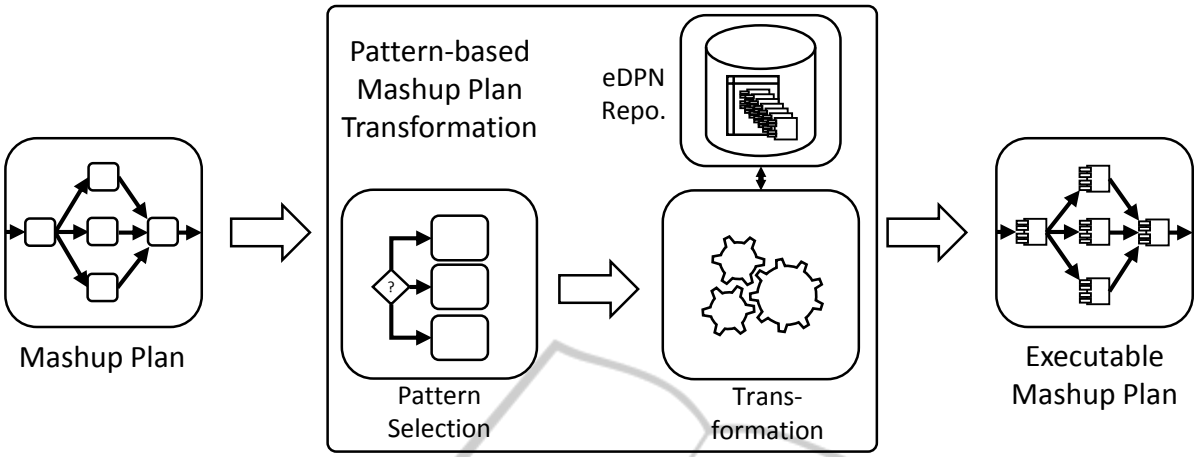


Figure 4: Mashup Plan Transformation Components.

non-technical users to model mashup scenarios they are interested in.

5 MASHUP PLAN TRANSFORMATION AND EXECUTION

The proposed approach to a modeling of Mashup Plans needs to be complemented by a transformation of the non-technical, domain-specific Mashup Plans into an efficiently executable model. Figure 4 shows this transformation. In the following, the transformation result is referred to as *executable Mashup Plan*. An executable Mashup Plan is also in the format of a directed graph, exclusively containing *executable data processing nodes (eDPN)* as nodes and the data and control flow between them as edges.

An eDPN represents an implementation, i.e., a piece of code, to be invoked (e.g., a Java Webservice) that executes data extraction (i.e., serves as data source adapter), data processing, e.g., implementing filter or join operations, or data storage operations. Information about the eDPN access (e.g., a Web Service URL) and their input parameters are described by the *eDPN Repository*, which is also implemented as a cloud-ready service. We recommend using existing data flow languages for the executable description. This prevents efforts for the creation of a self-made format and, as a consequence, a self-made execution engine (cf. Section 5.3).

Besides traditional workflow languages and engines (e.g., BPEL and ApacheODE), event processing engines such as CEP-Esper[5] or other data processing engines such as Node-RED[6] are also highly suitable for the definition and execution of executable Mashup Plans. Note that we are not restricted to a specific technical execution language at the execution level since we generate the right execution plans depending on the respective use case, i.e., on the transformation patterns.

5.1 Transformation Patterns

The transformation level allows the selection of transformation patterns that contain additional information about the scenario the mashup is executed in. As mentioned in (Daniel and Matera, 2014), different implementations are necessary for different mashup scenarios. Note that a detailed definition of transformation patterns as well as the creation of a transformation pattern catalog are part of our future work.

The following transformation patterns could, e.g., be selected: the transformation patterns *Time-Critical Mashup* or *Real-Time Mashup* define scenarios in which the run time of the mashup is an important factor. It is crucial that the corresponding implementation is chosen to be efficient rather than being robust. The transformation pattern *Robust Mashup* requires a robust execution, of course, the time-critical and robust transformation patterns cannot be both annotated to a Mashup Plan. A robust mashup has to be highly-available and has to offer exception handling as well as data persistence. The transformation pattern *Big Data Mashup* requires that the corresponding implementation has to support the processing of huge data sets in reasonable runtime. Note that more than one transformation pattern can be annotated to a Mashup Plan, however, some of these transformation patterns

[5]http://www.espertech.com/

[6]http://www.nodered.org/

cannot be combined. In future work, we detail on this approach.

The implementation, e.g., the used data flow language and execution engine is chosen according to the selected transformation patterns (cf. Section 5.2).

5.2 Mashup Plan Transformation

During the mashup plan transformation, first the transformation pattern is selected (e.g., "Robust Mashup", "Time-Critical Mashup"). The selection can be done statically, e.g., for all Mashups within a domain or it can be done for each mashup transformation individually by any user.

Afterwards, the DSDs and DPDs of the Mashup Plan are transformed to eDPNs that can be executed using a suitable execution engine. We use *data source adapters*, i.e., eDPNs implementing the extraction of data from given data sources. In the first step of the transformation, data source adapter eDPNs for each of the artifact-centric DSDs of the Mashup Plan are inserted into the executable model. As discussed in Section 4.2, a mapping of such artifact-centric descriptions to adapter eDPNs can be done, e.g., according to Sun et al. (Sun et al., 2014). By doing so, the domain-specific artifact models, e.g., an enterprise information system, are mapped to technical data structures such as a database table, a file-based storage or an unstructured text document. To support various kinds of data structures, the eDPN repository describes several data source adapters, each coping with different technical data structures. During the transformation, adapters suitable for given data structures are selected and inserted into the executable model. Next, eDPNs are inserted for the DPDs modeled in the Mashup Plan. Each DPD can be realized by an arbitrary number of eDPNs with a predefined execution order, depending on its context and the data to be processed. The corresponding eDPNs are taken from the eDPN repository. A mapping of DPDs to eDPNs through several abstraction levels could, e.g., be realized by a rule-based approach, as described by Reimann et al. (Reimann et al., 2014) or by Falkenthal et al. (Falkenthal et al., 2014). In our approach, the inserted eDPNs depend not only on their parameterization, but also on the position of the corresponding DPD in the Mashup Plan, i.e., on its predecessors and successors. In other words, the mapping of DPDs to eDPNs has to consider the given context. Furthermore, the mapping has to consider the selected transformation patterns. In our prototype, we implemented a mapping of two different transformation patterns to corresponding implementations: (i) implementing the *"Robust Mashup"* transformation pattern by realizing the executable model as workflow containing web service invocations (the eDPNs) as workflow steps and (ii) implementing the *"Time-Critical Mashup"* transformation pattern using the data processing engine Node-RED by implementing adapter eDPNs as HTTP REST resource calls and executable DPDs as JavaScript nodes (cf. Section 6). In the future, we will introduce further transformation patterns, e.g., suitable for CEP engines.

5.3 Mashup Plan Execution

After the Mashup Plan has been transformed into an executable one, it is executed by a suitable execution engine. We use a scalable, cloud-based execution engine like Node-RED or a workflow engine as described by Vukojevic-Haupt et al. (Vukojevic-Haupt et al., 2013) to process the executable Mashup Plan. An execution engine is a data flow-, workflow- or event processing-engine that invokes eDPNs in the order defined in the executable model. Furthermore, the execution engine controls the eDPN orchestration, data transfer and error handling. The eDPNs that are invoked by the execution engine have to be up and running, so they can be accessed efficiently during execution. We use a cloud-based, scalable runtime environment (e.g., a cloud-based web server), hosting the executable implementations of all available eDPNs. The input parameters and the paths to the implementations are described by the eDPNs. Their concrete implementation as well as the used execution engine depends on the selected transformation patterns. Using the OASIS standard TOSCA (OASIS, 2013) and the results of our previous work (Hirmer et al., 2014), we are able to deploy and execute the needed components (e.g., a workflow engine, a web server etc.) for each transformation pattern implementation automatically in a cloud environment. This enables a high degree of flexibility regarding the usage of different implementations, depending on the use case scenario.

6 PROTOTYPICAL IMPLEMENTATION AND EVALUATION

We implemented the *Time-Critical Mashup* and the *Robust Mashup* transformation patterns using different execution models and engines to prove the feasibility of our approach. These implementations are based on the motivating scenario described in Section 1. A paper describing the second transformation pattern implementation is – among other concepts – published in (Hirmer et al., 2015). In this section, we describe

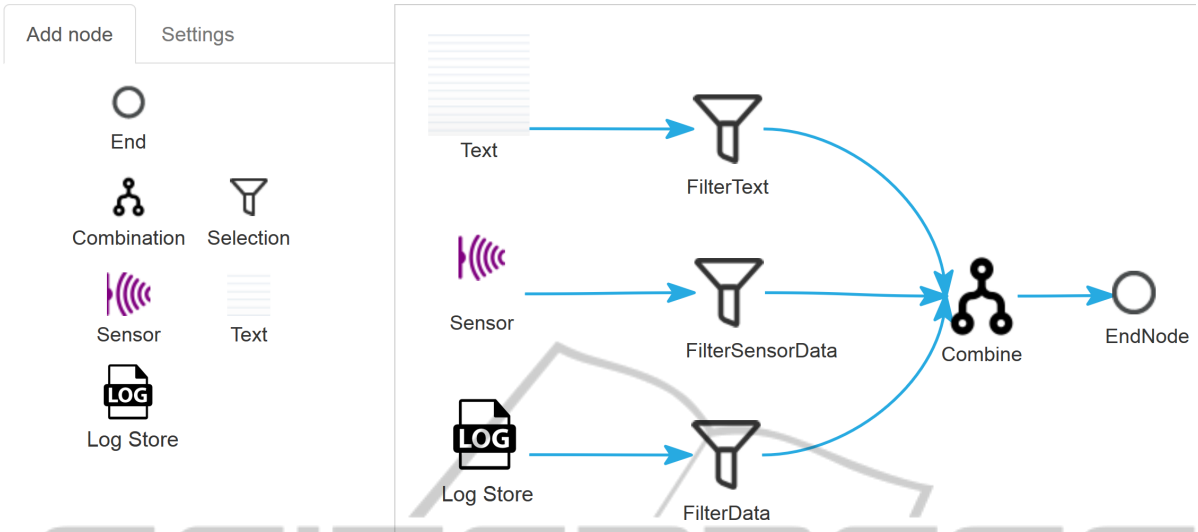


Figure 5: Mashup Plan Graphical Modeling Tool.

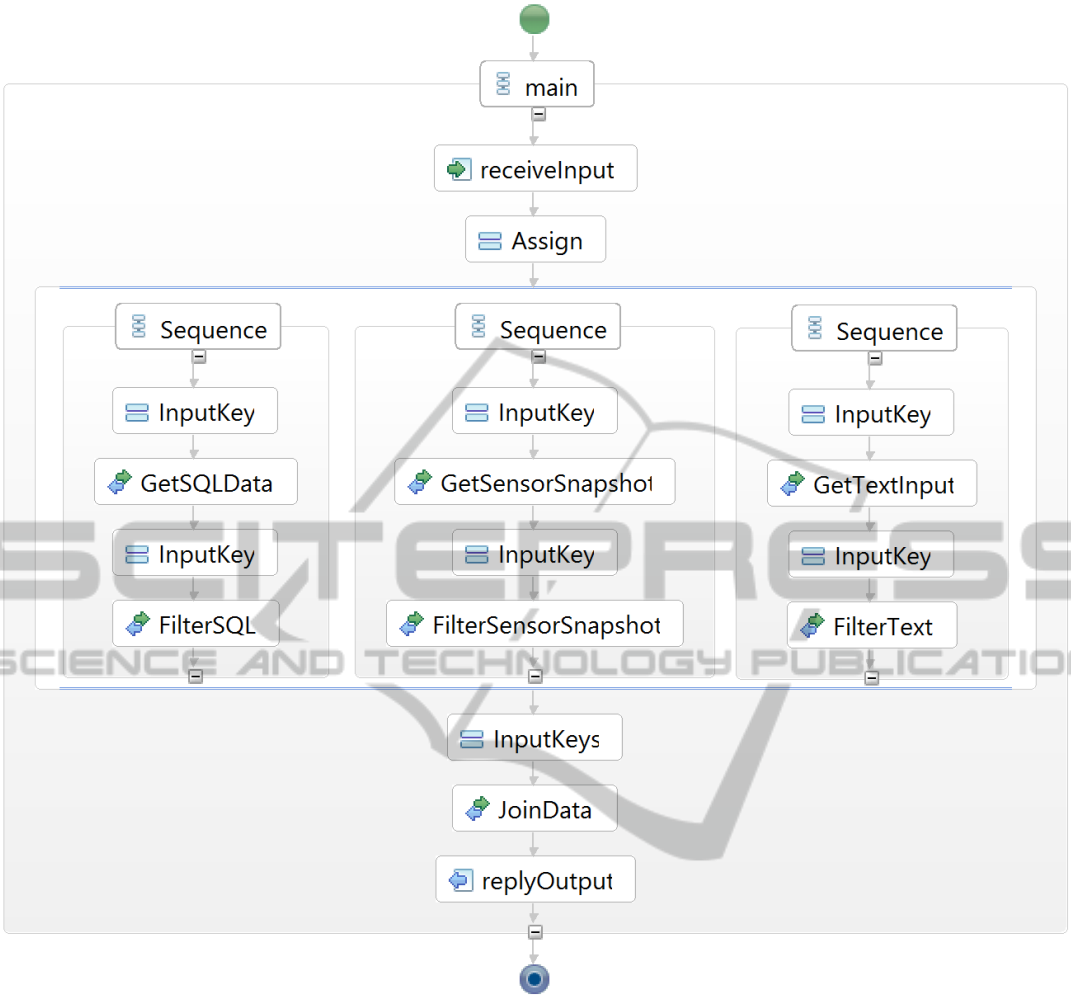
the implementation at modeling level. The transformation and execution levels are described for each implemented transformation pattern, separately.

A domain-specific Mashup Plan is realized in XML. For the modeling, we implemented a web-based, graphical modeling service based on Java, JavaScript and the JavaScript library AlloyUI[7], which is hosted by the platform-as-a-service provider IBM BlueMix[8] (cf. Figure 5). The DSD and DPD repositories have been implemented using MongoDB[9] key-value store services, containing an initial set of DSDs and DPDs. As data sources, we support structured MySQL[10] databases, unstructured text input feeds and sensor data. As Data Processing Descriptions, we support data combination and data selection DPDs introduced in Section 4.3.

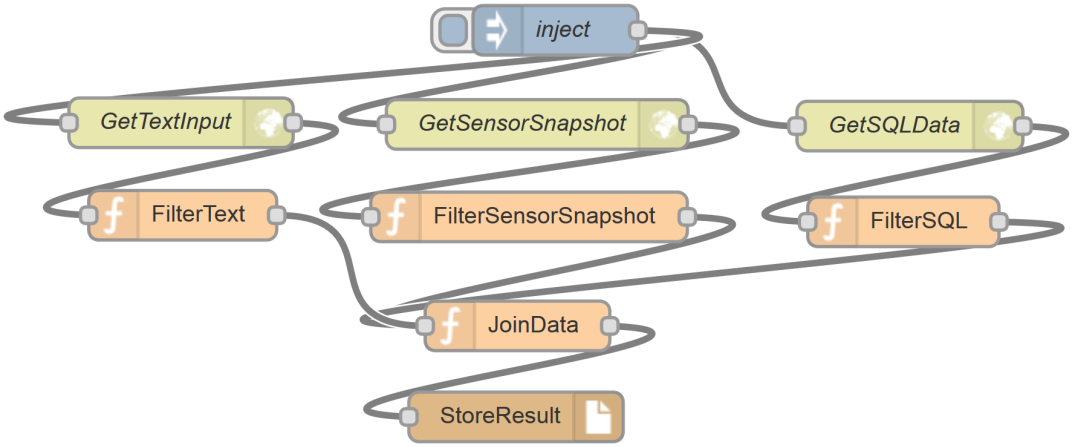
In this scenario, the data sources could, e.g., be used to compare log information stored in a database with corresponding text messages and sensor data output. Furthermore, the transformation patterns "Robust Mashup" or "Time-Critical" Mashup can be selected at transformation level. After modeling, the transformation can be invoked through the modeler's UI. In the following, we introduce the implementations of the "Robust Mashup" and "Time-Critical" transformation patterns as shown in Figure 6(a) and Figure 6(b), respectively.

Transformation Pattern 1 – Robust Mashup: For this transformation pattern, the transformation of the Mashup Plan to an executable model is realized by BPEL workflows (cf. Figure 6(a)), executed by the ApacheODE[11] workflow engine to ensure robustness. We used workflow technologies in this scenario due to the compliance with business requirements such as logging and exception handling. Furthermore, by using this transformation pattern we define that the mashup is not time-critical so the overhead produced by the engine is not relevant. The components of this implementation have been implemented as independent cloud services using IBM BlueMix. We use a cloud service, implementing the transformation of the Mashup Plan to a BPEL[12] workflow description. The workflow transformation so far uses a hard-coded DSD/DPD-to-eDPN mapping implemented in Java. The eDPNs to be executed are implemented as Java web services. We further implemented a service registry component using a MongoDB key-value store service that contains the Web Service Description Language (WSDL)[13] files of all implemented web services. These WSDL files are used to create BPEL service invocation nodes, i.e., the eDPNs, for the DSDs and DPDs of the Mashup Plan, which are executed when executing the workflow. We implemented data source adapter web services for the MySQL, sensor and text data sources as well as web services for the data combination and data selection DPDs. After the workflow is created, we

[7] http://www.alloyui.com/
[8] http://www.bluemix.net
[9] https://mms.mongodb.com/
[10] http://www.mysql.de/
[11] http://ode.apache.org/
[12] https://oasis-open.org/committees/wsbpel/
[13] http://w3.org/TR/wsdl



(a) Executable Mashup Plan as BPEL Workflow



(b) Executable Mashup Plan as Node-RED Flow

Figure 6: Different Implementations of the Same Executable Mashup Plan.

Table 1: Runtime Measurements.

Transformation Pattern	Transformation Time ∅	Deployment Time ∅	Execution Time ∅
Robust	284,4 ms	193,8 ms	2382 ms
Time-Critical	215,8 ms	134,6 ms	5,6 ms

execute it using the ApacheODE workflow engine. Table 1 shows the runtime measurements of this implementation.

Transformation Pattern 2 – Time-Critical Pattern: In this implementation, we used the same Mashup Plan as in the first implementation, however, we annotated it with the *Time-Critical Mashup* transformation pattern, i.e., we define real-time requirements. The Mashup Plan is mapped to an executable model based on JSON that is executed in the event processing engine Node-RED (cf. Figure 6(b)). The data is provided by REST resources that can be accessed through HTTP by the Node-RED processing nodes, i.e., the eDPNs for the DSDs. Furthermore, the DPDs are transformed to JavaScript nodes (the eDPNs) that are executed in the Node-RED engine respective to the order defined in the Mashup Plan. Table 1 shows the runtime measurements of this implementation. As shown in this table, the *Time-Critical Mashup* transformation pattern leads to a highly improved runtime, however, it lacks robustness.

Comparison of the Implementations: As already mentioned, we created two implementations focusing on different aspects. The first one implements a robust data mashup that copes with occurring errors in an effective manner using a workflow engine that is executing BPEL workflows. The second implementation enables processing very time-critical Mashup scenarios, e.g., in the Internet of Things area as described by (Hirmer et al., 2015). As shown in Table 1, transformation time and deployment time only show slight differences, which are mainly caused by internal data structures for representing the process models. However, the execution time measurements of these implementations differ greatly.

The robust execution runs a lot slower due to the heavy-weight workflow engine that is being used. The overhead produced by this engine mainly originates from a more complex implementation of the workflow navigation and from additional features enabling a robust workflow execution, such as auditing or workflow compensation. Furthermore, the engine requires the modeled data operations to be implemented by external Java-based web services, which leads to lots of communication overhead.

In contrast, the runtime of the time-critical implementation is significantly faster. This is enabled by the Internet of Things runtime environment Node-RED that is based on NodeJS, a platform made for efficient, data-intensive applications[14]. The extraordinary runtime of 5,6 ms is achieved through a more lightweight approach of workflow navigation that omits features for a robust workflow execution. In addition, Node-RED offers integrated and tailor-made data processing operators, which are much faster and prevent unnecessary communication overhead.

7 RELATED WORK

In general, other integration approaches, such as ETL processes, cannot cope with the requirements of the introduced scenario (cf. Section 1) due to several limitations. Firstly, ETL processes are very complex, hence difficult to scale, and require a huge amount of technical expertise for their creation, thus, cannot support an ad-hoc integration by only domain-experts. Secondly, ETL processes are static and lack flexibility and dynamics. The goal of this paper is not to substitute ETL processes, however, the realization of ad-hoc integration scenarios as presented in this paper, needs a more flexible approach.

In the following, we discuss related work that has been published: Soi et al. (Soi et al., 2014) mention the complexity and huge time-consumption faced during the creation of mashup platforms and the need for easier mashup development. By introducing the Mashup Development Kit (MDK), containing a new mashup language, the development of mashup platforms should be eased. Furthermore, De Vrieze et al. (de Vrieze et al., 2011) describe how to build enterprise mashups in a service-oriented way, using web services and workflow engines for their execution. Finally, Tietz et al. (Tietz et al., 2011) present a task-based approach for data mashups. In their paper, data mashups are executed by the consecutive execution of tasks, i.e., data operations. In contrast to our approach, these approaches do not focus on reducing the complexity of modeling data mashups in order to make it suitable for domain-experts. In our approach, we take the abstraction one step further by introducing domain-specific Mashup Plans, which are suitable for domain-experts. In addition, our concepts abstract from implementation details; this enables using differ-

[14]https://nodejs.org/

ent implementations, suitable for the respective scenarios. In other mashup approaches, the implementation is static and only fits a predefined scenario, i.e., it is not generic enough. Furthermore, through a cloud-ready approach, we can provide availability and scalability of our solution, which is an important factor, especially for coping with large volumes. Our approach can provide these features due to the provisioning of our solution in the cloud, using independently scalable cloud services. Finally, our solution enables extensibility, i.e., it is not bound to a fixed set of data sources and data operations. We enable this through a generic approach using extensible repositories and a flexible, transformation pattern-based transformation.

Wieland et al. (Wieland et al., 2015) present a similar approach to Mashup Plans and their transformation by introducing *Situation Templates*, a model for the integration of sensor data used for situation recognition in so called smart environments. However, in this approach, the focus is on integrating sensor data, i.e., other data sources such as relational databases or text sources are not supported.

8 SUMMARY AND FUTURE WORK

In this paper, we introduced extended techniques for flexible modeling and execution of data mashups. We enable the non-technical modeling and execution of ad-hoc integration scenarios by domain-experts. As described in the introduction, our goals were (i) the support of different use cases and scenarios through a generic mashup approach, (ii) the flexible, ad-hoc data integration by domain-experts, (iii) exploiting heterogeneous, i.e., structured and unstructured data sources, (iv) the creation of a scalable, stable and reusable solution, as well as (v) the dynamic (un-)tethering of data sources. The first goal (i) was realized by the introduction of transformation patterns that enable flexibility regarding different scenarios by a pattern-based transformation implementation selection. We enabled (ii) by introducing the modeling level, on which domain-experts can create their own integration scenarios using a domain-specific model. (iii) was achieved by the introduction of extensible DSD and DPD repositories. As a consequence, the support for heterogeneous data sources – either structured or unstructured – can be provided by technical experts, implementing these components, so the user can use them in an abstracted format themselves. We further enabled (iv), by creating a stable, reusable solution through the subdivision into stateless, scalable cloud services that can be managed independently. Finally, the last goal (v) is supported by the re-modeling and re-execution of Mashup Plans enabled by domain-specific models.

In this paper, we introduced a first version of our approach. In the future, we will continue working on these concepts by introducing a transformation pattern catalog containing further transformation patterns and corresponding implementations. We will also consider privacy and accountability aspects in the future due to the provisioning of our solution in a cloud environment. Our approach towards privacy and accountability in Mashups will build on existing work (Mohammed et al., 2009), (Zou et al., 2013). Furthermore, we will improve the domain-specific Mashup Plan model by introducing modeling patterns.

ACKNOWLEDGEMENT

This work is supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) within the Cluster of Excellence Simulation Technology (EXC 310/1) and within the project SitOPT (Grant 610872). We would like to thank the group of Prof. Frank Leymann of the Institute of Architecture of Application Systems[15] for their support and cooperation in these projects as well as the reviewers for their helpful comments.

REFERENCES

Binz, T., Breitenbücher, U., Kopp, O., and Leymann, F. (2014). *TOSCA: Portable Automated Deployment and Management of Cloud Applications*, pages 527–549. Advanced Web Services. Springer, New York.

Cohn, D. et al. (2009). Business artifacts: A Data-centric Approach to Modeling Business Operations and Processes. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*.

Daniel, F. and Matera, M. (2014). *Mashups - Concepts, Models and Architectures*. Data-Centric Systems and Applications. Springer.

de Vrieze, P. et al. (2011). Building enterprise mashups. *Future Generation Computer Systems*.

Falkenthal, M. et al. (2014). From Pattern Languages to Solution Implementations. In *Proceedings of the Sixth International Conferences on Pervasive Patterns and Applications (PATTERNS 2014), Venice, Italy*.

Hirmer, P., Breitenbücher, U., Binz, T., and Leymann, F. (2014). Automatic Topology Completion of TOSCA-based Cloud Applications. In *Proceedings des CloudCycle14 Workshops auf der 44. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*, volume 232 of *LNI*, pages 247–258, Bonn. Gesellschaft für Informatik e.V. (GI).

[15] http://www.iaas.uni-stuttgart.de/

Hirmer, P., Wieland, M., Schwarz, H., Mitschang, B., Breitenbücher, U., and Leymann, F. (2015). SitRS - A Situation Recognition Service based on Modeling and Executing Situation Templates. In *Proceedings of the 9th Symposium and Summer School On Service-Oriented Computing (SummerSOC)*.

Hoyer, V. et al. (2011). What Are the Business Benefits of Enterprise Mashups? In *System Sciences (HICSS), 44th Hawaii International Conference on System Sciences*.

Kassner, L. B. and Mitschang, B. (2015). MaXCept–Decision Support in Exception Handling through Unstructured Data Integration in the Production Context. An Integral Part of the Smart Factory. *In Proceedings of the 48th Hawaii International Conference on System Sciences*.

Künzle, V. et al. (2011). PHILharmonicFlows: Towards a Framework for Object-aware Process Management. *Journal of Software Maintenance and Evolution: Research and Practice*.

Meunier, R. (1995). The pipes and filters architecture. In *Pattern languages of program design*, pages 427–440. ACM Press/Addison-Wesley Publishing Co.

Mohammed, N. et al. (2009). Privacy-preserving Data Mashup. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*.

OASIS (2013). Topology and Orchestration Specification for Cloud Applications.

Reimann, P. et al. (2014). A Pattern Approach to Conquer the Data Complexity in Simulation Workflow Design. In *Proceedings of the 22nd International Conference on Cooperative Information Systems (CoopIS 2014), Amantea, Italy*.

Soi, S. et al. (2014). Conceptual Development of Custom, Domain-Specific Mashup Platforms. *ACM Trans. Web*.

Sun, Y. et al. (2014). Modeling Data for Business Processes. In *Proceedings of the 30th IEEE International Conference on Data Engineering (ICDE), Chicago, USA*.

Tietz, V. et al. (2011). Towards Task-based Development of Enterprise Mashups. In *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services*.

Vukojevic-Haupt, K. et al. (2013). On-demand Provisioning of Infrastructure, Middleware and Services for Simulation Workflows. In *Proceedings of the 6th IEEE International Conference on Service Oriented Computing & Applications (SOCA), Kauai, USA*.

Wieland, M., Schwarz, H., Breitenbücher, U., and Leymann, F. (2015). Towards Situation-Aware Adaptive Workflows. *In Proceedings of the 11th Workshop on Context and Activity Modeling and Recognition (COMOREA) @ IEEE Conference on Pervasive Computing (PerCom)*.

Zou, J. et al. (2013). Accountability in Enterprise Mashup Services. *Adv. Soft. Eng.*

Zweigle, O., Häussermann, K., Käppeler, U.-P., and Levi, P. (2009). Supervised Learning Algorithm for Automatic Adaption of Situation Templates Using Uncertain Data. In *Proceedings of the 2Nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, ICIS '09, pages 197–200, New York, NY, USA. ACM.